Dense Linear Algebra

Jakub Kurzak
kurzak@icl.utk.edu
http://www.icl.utk.edu/~kurzak/

Innovative Computing Laboratory
Electrical Engineering and Computer Science
University of Tennessee
Outline

- Legacy Software
  - BLAS
  - LINPACK
  - LAPACK
  - ScaLAPACK

- New Software
  - PLASMA & QUARK
  - DPLASMA & PaRSEC
LINPACK is a package of mathematical software for solving problems in linear algebra, mainly dense linear systems of linear equations.

The project had its origins in 1974

Written in Fortran 66

The project had four primary contributors: Jack Dongarra when he was at Argonne National Lab, Jim Bunch from the University of California-San Diego, Cleve Moler who was at New Mexico at that time, and Pete Stewart from the University of Maryland.

LINPACK as a software package has been largely superseded by LAPACK, which has been designed to run efficiently on shared-memory, vector supercomputers.
Computing in 1974

- High Performance Computers:
  - IBM 370/195, CDC 7600, Univac 1110, DEC PDP-10, Honeywell 6030
- Fortran 66
- Trying to achieve software portability
- Run efficiently
- BLAS (Level 1)
  - Vector operations
- Software released in 1979
  - About the time of the Cray 1
LU Factorization in LINPACK

1970's HPC of the day: vector architectures

- serial panel factorization
- Level 1 BLAS
- rank-1 update
- bulk-synchronous
Memory Hierarchy

- Processor
- CPU
  - CPU Cache
    - Level 1 (L1) Cache
    - Level 2 (L2) Cache
    - Level 3 (L3) Cache
- Physical Memory
  - Random Access Memory (RAM)
- Solid State Memory
  - Non-Volatile Flash-Based Memory
- Virtual Memory
  - File-Based Memory

- DDR, SD-RAM, DDR-SDRAM, RD-RAM
- EDO, SD-RAM, DDR-SDRAM, RD-RAM
- and More...
- SSD, Flash Drive
- Mechanical Hard Drives

Super Fast
Super expensive
Tiny capacity

Faster
Expensive
Small capacity

Fast
Priced reasonably
Average capacity

Average speed
Priced reasonably
Average capacity

Slow
Cheap
Large capacity

▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng
Level 1, 2 and 3 BLAS

Level 1 BLAS Matrix-Vector operations

**AXPY:**
\[ \alpha x + y \]

**DOT:**
\[ \alpha \]

2n FLOP
2n memory reference
RATIO: 1

Level 2 BLAS Matrix-Vector operations

**GEMV:**
\[ \alpha A x + y \]

2n² FLOP
n² memory references
RATIO: 2

Level 3 BLAS Matrix-Matrix operations

**GEHM:**
\[ \alpha A B + \beta C \]

2n³ FLOP
3n² memory references
RATIO: 2/3 n
Level 1, 2, and 3 BLAS
2.6 GHz Intel Sandy Bridge (single core)
Why is BLAS so important?

- Performance of lot of applications depends a lot on the performance of the underlying BLAS.

- Because the BLAS are efficient, portable, parallel, and widely available, they are commonly used in the development of high quality linear algebra software.
LAPACK

- Relies on BLAS
- Nearly 100% BLAS 3
  - linear systems of equations
  - least square problems
- About 50% BLAS 3
  - singular value decomposition
  - eigenvalue problems
- Error bounds for everything
  - single/double, real/complex
  - dense, symmetric, SPD, ...
- ca. 1700 routines
- ca. 500K lines of code
- ca. 500K lines of tests
LU Factorization in LINPACK

1970's HPC of the day: vector architectures

- serial panel factorization
- Level 1 BLAS
- rank-1 update
- bulk-synchronous
LU Factorization in LINPACK

1980's HPC of the day: cache-based architectures

- serial panel factorization
- Level 3 BLAS
- rank-k update
- bulk-synchronous
Typical LAPACK Routine

solve a system of linear equations using the LU factorization

subroutine dgesv( n, nrhs, A, lda, ipiv, b, ldb, info )

input:

\[ \begin{align*}
A & : n \times n \\
B & : n \times nrhs \\
\text{ipiv} & : n \\
\text{info} & : 1
\end{align*} \]

output:

\[ \begin{align*}
L & : n \times n \\
U & : n \times n \\
X & : nrhs \times n \\
\text{ipiv} & : n \\
\text{info} & : 1
\end{align*} \]

Solution of \( Ax = b \)
## LAPACK Functionality

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear system of equations</td>
<td>SV</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>LLS</td>
</tr>
<tr>
<td>Linear equality-constrained least squares problem</td>
<td>LSE</td>
</tr>
<tr>
<td>General linear model problem</td>
<td>GLM</td>
</tr>
<tr>
<td>Symmetric eigenproblems</td>
<td>SEP</td>
</tr>
<tr>
<td>Nonsymmetric eigenproblems</td>
<td>NEP</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>SVD</td>
</tr>
<tr>
<td>Generalized symmetric definite eigenproblems</td>
<td>GSEP</td>
</tr>
<tr>
<td>Generalized nonsymmetric eigenproblems</td>
<td>GNEP</td>
</tr>
<tr>
<td>Generalized (or quotient) singular value decomposition</td>
<td>GSVD (QSVD)</td>
</tr>
</tbody>
</table>
Parallelism in LAPACK

Parallelization of LU

Parallelize the update:
- Easy and done in any reasonable software.
- This is the 2/3n³ term in the FLOPs count.
- Can be done efficiently with LAPACK+multithreaded BLAS
ScaLAPACK

- Distributed memory
- Message Passing
  - clusters of SMPs
  - supercomputers
- Dense linear algebra
  - linear systems of equations
  - least square problems
  - singular value decomposition
  - eigenvalue problems
- Built on various modules
  - PBLAS, BLACS
- “object-oriented” FORTRAN 77 (array descriptor)
PBLAS

- Similar to the BLAS in functionality and naming.
- Built on the BLAS and BLACS.
- Provide global view of matrix:

  - CALL DGEXXX ( M, N, A( IA, JA ), LDA,... )

  - CALL PDGEXXX( M, N, A, IA, JA, DESCA,... )
**LAPACK**

[LAPACK] subroutine *dgesv* (n, nrhs, a(ia,ja), lda, ipiv, b(ib,jb), ldb, info)

**Input:**
- **A**: n x n matrix
- **B**: nrhs x n matrix
- **ipiv**: n vector

**Output:**
- **U**: n x n upper triangular matrix
- **L**: n x n lower triangular matrix
- **X**: nrhs x n matrix
- **ipiv**: n vector

**LAPACK Data layout**
- **info**
ScaLAPACK

[LAPACK] subroutine dgesv( n, nrhs, a(ia,ja), lda, ipiv, b(ib,jb), ldb, info )
[ScaLAPACK] subroutine pdgesv( n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info )

input:

\[
\begin{array}{ccc}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33} \\
\end{array}
\]

\[
\begin{array}{ccc}
B_{11} & \quad \quad & \quad \quad \\
B_{21} & \quad \quad & \quad \quad \\
B_{31} & \quad \quad & \quad \quad \\
\end{array}
\]

\[
\begin{array}{ccc}
\quad \quad & \quad \quad & ip_1 \\
\quad \quad & \quad \quad & ip_2 \\
\quad \quad & \quad \quad & ip_3 \\
\end{array}
\]

ScaLAPACK Data layout

info

output:

\[
\begin{array}{ccc}
U_{11} & U_{12} & U_{13} \\
L_{21} & U_{22} & U_{23} \\
L_{31} & L_{32} & U_{33} \\
\end{array}
\]

\[
\begin{array}{ccc}
X_{11} & \quad \quad & \quad \quad \\
X_{21} & \quad \quad & \quad \quad \\
X_{31} & \quad \quad & \quad \quad \\
\end{array}
\]

\[
\begin{array}{ccc}
\quad \quad & \quad \quad & ip_1 \\
\quad \quad & \quad \quad & ip_2 \\
\quad \quad & \quad \quad & ip_3 \\
\end{array}
\]

ScaLAPACK Data layout

info
ScaLAPACK
2D block cyclic data layout

Matrix point of view

Processor point of view

Matrix is MxN
Process grid is PxQ, P=2, Q=3
Blocks are MBxNB
**ScaLAPACK**

2D block cyclic data layout

<table>
<thead>
<tr>
<th>Matrix point of view</th>
<th>Processor point of view</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 4 0 2 4 0 2 4 0 2 4</td>
<td>0 0 0 2 2 2 2 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5 1 3 5</td>
<td>0 0 0 2 2 2 2 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4 0 2 4</td>
<td>0 0 0 2 2 2 2 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5 1 3 5</td>
<td>0 0 0 2 2 2 2 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5 1 3 5</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5 1 3 5</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 3 3 3 5 5 5</td>
</tr>
</tbody>
</table>
**ScaLAPACK**

2D block cyclic data layout

<table>
<thead>
<tr>
<th>Matrix point of view</th>
<th>Processor point of view</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Matrix view" /></td>
<td><img src="image2" alt="Processor view" /></td>
</tr>
</tbody>
</table>

The diagram illustrates the 2D block cyclic data layout in ScaLAPACK. The matrix is partitioned into blocks, and these blocks are distributed across processors. The matrix point of view shows the arrangement of elements in the matrix, while the processor point of view illustrates how these elements are distributed among processors.
ScaLAPACK

2D block cyclic data layout

Matrix point of view

Processor point of view
ScaLAPACK
2D block cyclic data layout
### ScaLAPACK

2D block cyclic data layout

<table>
<thead>
<tr>
<th>Matrix point of view</th>
<th>Processor point of view</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 0 2 2 2 2 2 2</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 0 2 2 2 2 2 2</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>0 0 0 4 4 4 4 4 4</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>0 0 0 4 4 4 4 4 4</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 3 3 3 3 3 3</td>
</tr>
<tr>
<td>1 3 5 1 3 5 1 3 5</td>
<td>1 1 1 5 5 5 5 5 5</td>
</tr>
<tr>
<td>0 2 4 0 2 4 0 2 4</td>
<td>1 1 1 5 5 5 5 5 5</td>
</tr>
</tbody>
</table>
ScaLAPACK
2D block cyclic data layout
ScaLAPACK
2D block cyclic data layout
ScaLAPACK
2D block cyclic data layout

Matrix point of view

Processor point of view

0 0 2 2
0 0 2 2
0 0 2 2

1 1 3 3
1 1 3 3
1 1 3 3

ScaLAPACK
2D block cyclic data layout
## LAPACK Functionality

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear system of equations</td>
<td>SV</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>LLS</td>
</tr>
<tr>
<td>Linear equality-constrained least squares problem</td>
<td>LSE</td>
</tr>
<tr>
<td>General linear model problem</td>
<td>GLM</td>
</tr>
<tr>
<td>Symmetric eigenproblems</td>
<td>SEP</td>
</tr>
<tr>
<td>Nonsymmetric eigenproblems</td>
<td>NEP</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>SVD</td>
</tr>
<tr>
<td>Generalized symmetric definite eigenproblems</td>
<td>GSEPN</td>
</tr>
<tr>
<td>Generalized nonsymmetric eigenproblems</td>
<td>GNENP</td>
</tr>
<tr>
<td>Generalized (or quotient) singular value decomposition</td>
<td>GSVD (QSVD)</td>
</tr>
</tbody>
</table>
# ScaLAPACK Functionality

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear system of equations</td>
<td>SV</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>LLS</td>
</tr>
<tr>
<td>Linear equality-constrained least squares problem</td>
<td>LSE</td>
</tr>
<tr>
<td>General linear model problem</td>
<td>GLM</td>
</tr>
<tr>
<td>Symmetric eigenproblems</td>
<td>SEP</td>
</tr>
<tr>
<td>Nonsymmetric eigenproblems</td>
<td>NEP</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>SVD</td>
</tr>
<tr>
<td>Generalized symmetric definite eigenproblems</td>
<td>GSEP</td>
</tr>
<tr>
<td>Generalized nonsymmetric eigenproblems</td>
<td>GNEP</td>
</tr>
<tr>
<td>Generalized (or quotient) singular value decomposition</td>
<td>GSVD (QSVD)</td>
</tr>
</tbody>
</table>
Parallelism in ScaLAPACK

Parallelization of QR Factorization

- Parallelize the update:
  - Easy and done in any reasonable software.
  - This is the $2/3n^3$ term in the FLOPs count.
  - Can be done efficiently with LAPACK+multithreaded BLAS

- Panel factorization
  - dgeqf2 + dlarft

- Update of the remaining submatrix
  - dlarfb

Fork - Join parallelism
Bulk Sync Processing
Dense Linear Algebra

legacy software libraries

- **LINPACK** (70's)
  - vector operations
  - Level 1 BLAS

- **LAPACK** (80's)
  - block operations
  - Level 3 BLAS

- **ScaLAPACK** (90's)
  - block cyclic data distribution
  - PBLAS
  - BLACS
  - MPI
Dense Linear Algebra
the rise of multicore
Dense Linear Algebra
legacy software libraries

what about ScaLAPACK?
Dense Linear Algebra
legacy software libraries

Cholesky Inversion (POTRF+POTRI), N=4000,
MKL 10,2, lapack 3.2.1, scalapack 1.8.0, AMD Istanbul
Eight hexa-core Opteron 8439 SE (codename Istanbul) processors 2.8 GHz

- Theoretical peak
- Perfect scalability
- MKL LAPACK
- Reference LAPACK
- ScaLAPACK
Dense Linear Algebra
emerging software solutions

PLASMA
- tile algorithms
- tile layout
- dynamic scheduling* (superscalar)

MAGMA
- block algorithms* (LAPACK)
- standard layout
- static scheduling*

DPLASMA
- tile algorithms
- tile layout
- dynamic scheduling (parametrized task graph)

HOMOGENEOUS MULTICORE

HYBRID:
MULTICORE + ACCELERATOR / MANYCORE / CO-PROCESSOR

LARGE DISTRIBUTED (WITH ACCELERATORS)

* mostly
Matrix Layout

tile data layout

(Sca)LAPACK
column-major

(D)PLASMA
column-column rectangular block

- cache & TLB efficiency (conflict misses, false sharing)
- messaging efficiency (zero-copy communication)
Matrix Layout

Contiguous memory region containing A11, A21, A12, A22

Each tile is contiguous (column-major)

Each panel is contiguous
Matrix Layout
layout conversion

(Sca)LAPACK
column-major

(D)PLASMA
column-column rectangular block

- in place (zero memory overhead)
- parallel (multithreaded)
- cache efficient
Matrix Layout
layout conversion

Column Major / Row Major

C|R C|R Rectangular Block

- fast translation between any layouts
- fast transposition
Algorithms

singular value decomposition – reduction to band

alternating steps of QR and LR reductions by tiles
Algorithms

singular value decomposition – band reduction

parallel, cache-efficient sweep down the band
Algorithms

singular value decomposition – two stage bidiagonal reduction

- **algorithm**
  - reduction to band + band reduction
  - find singular values
  - compute singular vectors

- **numerics**
  - same as LAPACK

- **performance**
  - ~10x if singular values only
  - ~2x if all singular vectors

- storing by tiles
- computing by tiles
- dynamic scheduling
Dataflow Scheduling
DAG construction / critical path

top-looking
left-looking
right-looking

Cholesky factorization
Dataflow Scheduling
DAG construction / critical path

LAUUM
triangular matrix multiply

in-place
out-of-place
Dataflow Scheduling

DAG construction / critical path

in-place

out-of-place

Cholesky-based matrix inversion
POTRF + TRTRI + LAUUM
Dataflow Scheduling

DAG construction / critical path

<table>
<thead>
<tr>
<th>In place, no interleaving</th>
<th>Out of place, no interleaving</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9t-6 = (3t-2) + (3t-3) + (3t-2)

6t-3 = (3t-2) + (2t-1) + t
## Dataflow Scheduling

**DAG construction / critical path**

<table>
<thead>
<tr>
<th>In place, no interleaving</th>
<th>Out of place, no interleaving</th>
<th>Out of place, interleaving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
9t-6 = (3t-2) + (3t-3) + (3t-2) \quad 6t-3 = (3t-2) + (2t-1) + t \quad 5t-2
\]

**Cholesky-based matrix inversion**

\[ \text{POTRF + TRTRI + LAUUM} \]
Dataflow Scheduling
DAG composition

Cholesky-based matrix inversion
POTRF + TRTRI + LAUUM
Dataflow Scheduling

DAG composition

Cholesky-based matrix inversion
POTRF + TRTRI + LAUUM
Math & CS resources

http://www.netlib.org/lapack/lawns/

LAWNs (LAPACK Working Notes)

LAWNS hibtex

lawn283 [pdf]
An Improved Parallel Singular Value Algorithm and Its Implementation for Multicore Hardware
by Azzam Haidar, Piotr Luszczek, Jakub Kurzak, and Jack Dongarra
UT-EECS-13-720 October 2013

lawn282 [pdf]
Designing LU-QR hybrid solvers for performance and stability
by Mathieu Faverge, Julien Herrmann, Julien Langou, Bradley Lowery, Yves Robert and Jack Dongarra
UT-EECS-13-719 October 2013

lawn281 [pdf]
Optimal Checkpointing Period: Time vs. Energy
by Guillaume Aupy, Anne Benoît, Thomas Herault, Yves Robert and Jack Dongarra
UT-EECS-13-718 October 2013

lawn280 [pdf]
On Algorithmic Variants of Parallel Gaussian Elimination: Comparison of Implementations in Terms of Performance and Numerical Properties
by Simplice Donfack, Jack Dongarra, Mathieu Faverge, Mark Gates, Jakub Kurzak, Piotr Luszczek, and Ichitaro Yamazaki
UT-CS-13-715 July 2013

lawn279 [pdf]
Transient Error Resilient Hessenberg Reduction on GPU-based Hybrid Architectures
by Yulu Jia, Piotr Luszczek, and Jack Dongarra
UT-CS-13-712 June 2013

questions?
PLASMA
software stack

PLASMA

core BLAS

CBLAS

LAPACKE

(C)LAPACK

BLAS

POSIX Threads
WinThreads

hwloc
FOR \( k = 0 \ldots \text{TILES-1} \)
\[
A[k][k] \leftarrow \text{DPOTRF}(A[k][k])
\]
FOR \( m = k+1 \ldots \text{TILES-1} \)
\[
A[m][k] \leftarrow \text{DTRSM}(A[k][k], A[m][k])
\]
FOR \( m = k+1 \ldots \text{TILES-1} \)
\[
A[m][m] \leftarrow \text{DSYRK}(A[m][k], A[m][m])
\]
FOR \( n = k+1 \ldots m-1 \)
\[
A[m][n] \leftarrow \text{DGEMM}(A[m][k], A[n][k], A[m][n])
\]

\[
\text{for } (k = 0; k < A.mt; k++) \{
\text{QUARK\_CORE\_dpotrf}(\ldots);
\text{for } (m = k+1; m < A.mt; m++) \{
\text{QUARK\_CORE\_dtrsm}(\ldots);
\}
\text{for } (m = k+1; m < A.mt; m++) \{
\text{QUARK\_CORE\_dsyrk}(\ldots);
\text{for } (n = k+1; n < m; n++) \{
\text{QUARK\_CORE\_dgemm}(\ldots)
\}
\}
\}
\]
superscalar scheduling
- serial code
- side-effect-free tasks
- dependency resolution

resolving data hazards
- read after write
- write after read
- write after write

similar approach
- StarPU from Inria
- SMPSs from Barcelona
- SuperGlue and DuctTEiP from Uppsala
- Jade from Stanford (historical)
- OpenMP
Performance Comparison

singular value decomposition
Performance Comparison

singular value decomposition
PLASMA / QUARK

resources

http://icl.cs.utk.edu/plasma/

http://icl.cs.utk.edu/quark/

questions?
Dataflow Execution

distributed memory
Dataflow Execution

distributed memory

A00 A01 A02 A03
A10 A11 A12 A13
A20 A21 A22 A23
A30 A31 A32 A33

local dependency
communication
DPLASMA software stack

DPLASMA

(geqrf, potrf, getrf, gesv, ...)

PLASMA Algorithms

DPLASMA

Tile Kernels

MAGMA

GPU Kernels

PaRSEC

Parallel Runtime Scheduling and Execution Control

BLAS

CUDA

hwloc

pthread

MPI
PaRSEC
parametrized task graphs

- **parametrized task graph**
  - symbolic DAG representation
  - problem size independent
  - completely distributed

- **runtime implementation**
  - data-driven execution
  - locality-aware scheduling
  - communication overlapping
**PaRSEC**
from serial code to PTG

```plaintext
FOR k = 0 .. SIZE-1
    A[k][k], T[k][k] <- DGEQRT(A[k][k])
FOR m = k+1 .. SIZE-1
    A[k][k], A[m][k], T[m][k] <- DTSQRT(A[k][k], A[m][k], T[m][k])
FOR n = k+1 .. SIZE-1
    A[k][n] <- DORMQR(A[k][k], T[k][k], A[k][n])
FOR m = k+1 .. SIZE-1
    A[k][n], A[m][n] <- DSSMQR(A[m][k], T[m][k], A[k][n], A[m][n])
```
PaRSEC
from serial code to PTG

dataflow analysis
FOR $k=0$ TO $N-1$

DGEQRT($i_{out}A_{kk}$)

FOR $n=k+1$ to $N$

DORMQR($in A_{kk}, inout A_{kn}$)

FOR $m=k+1$ to $N$

DTSQRT($i_{out}A_{kk}, inout A_{mk}$)

FOR $n=k+1$ to $N$

DTSMQR($in A_{mk}, inout A_{kn}, inout A_{mn}$)

serial code

DGEQRT, $kkk$

$1_{ARG} \leftarrow A_{k,k} \mid DTSMQR_{k,k,k-1}$

$1_{ARG} \Rightarrow DORMQR_{k,k+1..N,k}(\mathbb{I})$

$1_{ARG} \Rightarrow DTSQRT_{k+1,k,k}(\mathbb{I})$

$1_{ARG} \Rightarrow A_{k,k}(\mathbb{I})$

DORMQR, $knk$

$1_{ARG} \leftarrow DGEQRT_{k,k}(\mathbb{I})$

$2_{ARG} \leftarrow A_{k,n} \mid DTSMQR_{k,n,k-1}$

$2_{ARG} \Rightarrow DTSMQR_{k+1,n,k}$

$2_{ARG} \Rightarrow A_{k,n}$

DTSQRT, $mkk$

$1_{ARG} \leftarrow DGEQRT_{m-1,k,k}(\mathbb{I}) \mid DTSQRT_{m-1,k,k}(\mathbb{I})$

$1_{ARG} \Rightarrow DTSQRT_{m+1,k,k}(\mathbb{I}) \mid A_{k,k}(\mathbb{I})$

$2_{ARG} \leftarrow A_{m,k} \mid DTSMQR_{m,k,k-1}$

$2_{ARG} \Rightarrow DTSMQR_{m,k+1..N,k}$

$2_{ARG} \Rightarrow A_{m,k}$

DTSMQR, $mkn$

$1_{ARG} \leftarrow DTSQRT_{m,k,k}$

$2_{ARG} \leftarrow DORMQR_{m-1,n,k} \mid DTSMQR_{m-1,n,k}$

$2_{ARG} \Rightarrow DTSMQR_{m+1,n,k} \mid A_{n,k}$

$3_{ARG} \leftarrow A_{m,n} \mid DTSMQR_{m,n,k-1}$

$3_{ARG} \Rightarrow DGEQRT_{m,n,k+1} \mid DORMQR_{m,n,k+1} \mid$

$\Rightarrow DTSQRT_{m,n,k+1} \mid DTSMQR_{m,n,k+1}$

$\Rightarrow A_{m,n}$

PTG representation
Solving Linear Least Square Problem (DGEQRF)
60-node, 480-core, 2.27GHz Intel Xeon Nehalem, IB 20G System

THEORETICAL PEAK OF 4358.4 GFLOP/S

DPLASMA

ScaLAPACK
DPLASMA's Performance
distributed (multicore + accelerator)

Solving Hermitian Positive-Definite System (SPOTRF)
12-node, 96-core, 2.27GHz Intel Xeon Nehalem, IB 20G System
w/ 12-Tesla C2070 GPU

WEAK SCALING

PRACTICAL PEAK
(SGEMM PERFORMANCE)

DPLASMA
(WITH GPU)

ScalAPACK
(WITHOUT GPU)

TFLOP/S

NUMBER OF NODES

MATRIX SIZE (MNxN)

1 2 4 8 12
54k 76k 108k 152k 176k
DPLASMA's Performance
distributed (multicore + multiple accelerators)
DPLASMA / PaRSEC

resources

http://icl.cs.utk.edu/dplasma/

http://icl.cs.utk.edu/parsec/